Paleti Krishnasai
CED18I039

Operating Systems - FORK_EXEC Assignment-4

1.) Test drive a C program to test drive ORPHAN and ZOMBIE processes

Theory :

Orphan Process : A process in which the parent / calling process is terminated before the corresponding child process. [parent dying before their children in real world] , this happens when the calling process does not wait is not properly synchronized.

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
   pid_t pid = fork();

   if(pid > 0)
   {
       printf("Parent Process\n");
   }
   else if(pid == 0)
   {
       sleep(30);
       printf("Child Process\n");

   }
return 0;
```

```
}
```

```
paleti@paletil:~/OS_LAB/fork_3$ make orphan
cc     orphan.c    -o orphan
paleti@paletil:~/OS_LAB/fork_3$ ./orphan
Parent Process
paleti@paletil:~/OS_LAB/fork_3$ Child Process
^C
paleti@paletil:~/OS_LAB/fork_3$
```

Zombie Process : A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table. In the following code, the child finishes its execution using exit() system call while the parent sleeps for 50 seconds, hence doesn't call wait() and the child process's entry still exists in the process table.

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
   pid_t pid = fork();
   if (pid > 0)
   {
       sleep(50);
       printf("Parent Process\n");
   }
   else if(pid == 0)
   {
       printf("Child Process\n");
```

```
        exit(0);
    }

    return 0;
}
```

```
paleti@paletil:~/OS_LAB/fork_3$ make zombie
cc     zombie.c   -o zombie
paleti@paletil:~/OS_LAB/fork_3$ ./zombie
Child Process
Parent Process
paleti@paletil:~/OS_LAB/fork_3$
```

(there is a considerable delay before **Parent Process** is printed**)**

2.) Develop a multiprocessing version of MERGE or QUICK sort

Theory :

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted subarrays into one

Merge sort is a stable sorting algorithm.

Here we split the workload into processes to handle the splitting the two subhalfs.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
```

Paleti Krishnasai
CED18I039

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

void merge(int arr[], int l, int m, int r)
{
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;

   int L[n1], R[n2];

   for (i = 0; i < n1; i++)
       L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
       R[j] = arr[m + 1 + j];

   i = 0;
   j = 0;
   k = l;
   while (i < n1 && j < n2)
   {
       if (L[i] <= R[j])
       {
           arr[k] = L[i];
           i++;
       }
       else
       {
           arr[k] = R[j];
           j++;
       }
       k++;
   }

   while (i < n1)
   {
```

```
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l ,int r)
{
    if(l < r)
    {
        int m = l + (r-l)/2;
        if(vfork() == 0)
        {
            mergeSort(arr,l,m);
            exit(0);
        }
        else
        {
            mergeSort(arr,m+1,r);
            merge(arr,l,m,r);
        }

    }
}

void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
```

Paleti Krishnasai
CED18I039

```
   printf("\n");
}

int main()
{
   int range;
   printf("Enter the size of the input : ");
   scanf("%d",&range);
   int arr[range];
   printf("Enter the input : \n");
   for(int i=0;i<range;i++)
   {
       scanf("%d",&arr[i]);
   }

   printf("%d\n",range);
   printf("Given array is \n");
   printArray(arr, range);

   mergeSort(arr, 0, range - 1);

   printf("\nSorted array is \n");
   printArray(arr, range);
   return 0;
}
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/fork_3$ ./mergesort
Enter the size of the input : 6
Enter the input :
1 1 0 0 3 2
6
Given array is
1 1 0 0 3 2

Sorted array is
0 0 1 1 2 3
paleti@paletil:~/OS_LAB/fork_3$ ./mergesort
Enter the size of the input : 10
Enter the input :
6 9 99 25 36 14 0 0 2 2
10
Given array is
6 9 99 25 36 14 0 0 2 2

Sorted array is
0 0 2 2 6 9 14 25 36 99
```

Theory:

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Unlike merge sort , quick sort is **not** stable.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
```

Paleti Krishnasai
CED18I039

```c
#include <unistd.h>

void quicksort(int number[], int first, int last)
{
   int i, j, pivot, temp;

   if (first < last)
   {
       pivot = first;
       i = first;
       j = last;

       while (i < j)
       {
           while (number[i] <= number[pivot] && i < last)
               i++;
           while (number[j] > number[pivot])
               j--;
           if (i < j)
           {
               temp = number[i];
               number[i] = number[j];
               number[j] = temp;
           }
       }

       temp = number[pivot];
       number[pivot] = number[j];
       number[j] = temp;

       if (vfork() == 0)
       {
           quicksort(number, first, j - 1);
           exit(0);
       }
       quicksort(number, j + 1, last);
   }
```

```
}

void printArray(int arr[], int size)
{
   int i;
   for (i = 0; i < size; i++)
       printf("%d ", arr[i]);
   printf("\n");
}

int main()
{

   int n;
   printf("Enter size of array:-\n");
   scanf("%d", &n);

   int arr[n];
   printf("Enter array elements:-\n");
   for (int i = 0; i < n; i++)
   {
       scanf("%d", &arr[i]);
   }

   printf("Given array is \n");
   printArray(arr, n);

   quicksort(arr, 0, n - 1);

   printf("\nSorted array is \n");
   printArray(arr, n);
   return 0;
}
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/fork_3$ ./quicksort
Enter size of array:-
5
Enter array elements:-
5 5 4 3 2
Given array is
5 5 4 3 2

Sorted array is
2 3 4 5 5
paleti@paletil:~/OS_LAB/fork_3$ ./quicksort
Enter size of array:-
10
Enter array elements:-
99 99 65 323 465 8 2 1 0 0
Given array is
99 99 65 323 465 8 2 1 0 0

Sorted array is
0 0 1 2 8 65 99 99 323 465
paleti@paletil:~/OS_LAB/fork_3$
```

3.) Develop a C program to count the maximum number of process that can be created using fork call

Theory :
Uncontrolled forking to check the number of process created.
Vfork used for data sharing.

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Paleti Krishnasai
CED18I039

```c
int main()


{
   pid_t pid;
   int count=0;
   while(1)
   {
       pid = vfork();
       if (pid == 0)
       {
           count++;
           exit(0);
       }
       else if(pid == -1)
       {
           printf("Max Processes allowed :%d\n",count);
           exit(-1);
       }

   }
   return 0;
}
```

```
paleti@paletil:~/OS_LAB/fork_3$ ./processcount
Max Processes allowed :19572
paleti@paletil:~/OS_LAB/fork_3$ ./processcount
Max Processes allowed :19571
paleti@paletil:~/OS_LAB/fork_3$ ./processcount
Max Processes allowed :19572
paleti@paletil:~/OS_LAB/fork_3$
```

4.) Develop your own command shell [say mark it with @] that accepts user commands(system or user binaries), executes the commands and returns the prompt for further user interaction. Also extend this to support a history feature.

Paleti Krishnasai
CED18I039

Code : without IPC.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

void history(char his[],char cmd[])
{
   strcat(his,"\n");
   strcat(his,cmd);

}



int main()
{

   char his[1000]="";//empty
   char cmd[100]={0};//initialize to 0
   char temp[100][100]={0};
   int len_temp = 0;


   while(1)
   {
       printf("%s","\npaleti@paletil:~$ ");
       scanf("%[^\n]%*c",cmd);
       strcpy(temp[len_temp],cmd);
       len_temp++;
       history(his, cmd);
       if (strcmp(cmd, "quit") == 0)
       {
           break;
```

```
        }
        char arg[10][100]={0};
        int argc=0;
        int count =0;
        for(int i=0;i<strlen(cmd);i++)
        {
            if(cmd[i]==' ')
            {
                argc++;
                count=0;
            }
            else
            {
                arg[argc][count++] = cmd[i];
            }
        }
        char *argv[10]={0};
        int k =0;
        for(k=0;k<=argc;k++)
             argv[k]=arg[k];
         argv[k]=NULL;

        pid_t pid ;
        pid = fork();
        if(pid == 0)
        {
            if(!(strcmp(cmd,"history")))
            printf("%s\n",his);
            else if (cmd[0]=='!')
            {
                //strcat()
                int vck = atoi(&cmd[1]);
                for(int i=vck-1;i>-1;i--)
                {
                    printf("%s\n",temp[i]);
                }
```

Paleti Krishnasai
CED18I039

```
            }
            else
            {
                if(execvp(*argv,argv)<0)
                printf("Invalid Command!!!\n");
            }
            exit(0);

        }
        else
        wait(NULL);

    }
}
```

```
paleti@paletil:~/OS_LAB/fork_3$ ./cmd

paleti@paletil:~$ ls
cmd    cmd_2.c  fork-3     histogram.c  magiccheck.c   magicgenerate.c       matrixmultiplication.c  mergesort.c  orphan.c      processcount.c  quicksort.c  zombie.c
cmd_2  cmd.c    histogram  magiccheck   magicgenerate  matrixmultiplication  mergesort               orphan       processcount  quicksort       zombie

paleti@paletil:~$ ls -l
total 1408
-rwxrwxr-x 1 paleti paleti   17296 Oct  4 23:51 cmd
-rwxrwxr-x 1 paleti paleti   17384 Oct  4 22:04 cmd_2
-rw-rw-r-- 1 paleti paleti    1646 Oct  4 22:04 cmd_2.c
-rw-rw-r-- 1 paleti paleti    1697 Oct  4 23:51 cmd.c
-rw-rw-r-- 1 paleti paleti 1166882 Sep 16 12:13 fork-3
-rwxrwxr-x 1 paleti paleti    8624 Sep 22 17:19 histogram
-rw-rw-r-- 1 paleti paleti    1101 Sep 22 17:19 histogram.c
-rwxrwxr-x 1 paleti paleti   16960 Oct  4 22:52 magiccheck
-rw-rw-r-- 1 paleti paleti    2028 Oct  4 22:52 magiccheck.c
-rwxrwxr-x 1 paleti paleti   21432 Oct  4 22:34 magicgenerate
-rw-rw-r-- 1 paleti paleti    9375 Oct  4 23:19 magicgenerate.c
-rwxrwxr-x 1 paleti paleti   16976 Oct  4 20:40 matrixmultiplication
-rw-rw-r-- 1 paleti paleti    1505 Oct  4 20:39 matrixmultiplication.c
-rwxrwxr-x 1 paleti paleti   17064 Oct  4 17:34 mergesort
-rw-rw-r-- 1 paleti paleti    1675 Oct  4 17:34 mergesort.c
-rwxrwxr-x 1 paleti paleti   16776 Oct  3 17:49 orphan
-rw-rw-r-- 1 paleti paleti     345 Oct  3 17:48 orphan.c
-rwxrwxr-x 1 paleti paleti   16784 Oct  4 18:09 processcount
-rw-rw-r-- 1 paleti paleti     479 Oct  4 18:09 processcount.c
-rwxrwxr-x 1 paleti paleti   17040 Oct  4 17:58 quicksort
-rw-rw-r-- 1 paleti paleti    1428 Oct  4 17:57 quicksort.c
-rwxrwxr-x 1 paleti paleti   16816 Oct  3 17:58 zombie
-rw-rw-r-- 1 paleti paleti     370 Oct  3 17:56 zombie.c

paleti@paletil:~$ dir
dir: cannot access '': No such file or directory

paleti@paletil:~$ dir
cmd    cmd_2.c  fork-3     histogram.c  magiccheck.c   magicgenerate.c       matrixmultiplication.c  mergesort.c  orphan.c      processcount.c  quicksort.c  zombie.c
cmd_2  cmd.c    histogram  magiccheck   magicgenerate  matrixmultiplication  mergesort               orphan       processcount  quicksort       zombie
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~$ dir
dir: cannot access '': No such file or directory

paleti@paletil:~$ dir
cmd    cmd_2.c  fork-3     histogram.c  magiccheck.c   magicgenerate.c       matrixmultiplication.c  mergesort.c  orphan.c      processcount.c  quicksort.c  zombie.c
cmd_2  cmd.c    histogram  magiccheck   magicgenerate  matrixmultiplication  mergesort               orphan       processcount  quicksort       zombie

paleti@paletil:~$ ls -lh
total 1.4M
-rwxrwxr-x 1 paleti paleti  17K Oct  4 23:51 cmd
-rwxrwxr-x 1 paleti paleti  17K Oct  4 22:04 cmd_2
-rw-rw-r-- 1 paleti paleti 1.7K Oct  4 22:04 cmd_2.c
-rw-rw-r-- 1 paleti paleti 1.7K Oct  4 23:51 cmd.c
-rw-rw-r-- 1 paleti paleti 1.2M Sep 16 12:13 fork-3
-rwxrwxr-x 1 paleti paleti 8.5K Sep 22 17:19 histogram
-rw-rw-r-- 1 paleti paleti 1.1K Sep 22 17:19 histogram.c
-rwxrwxr-x 1 paleti paleti  17K Oct  4 22:52 magiccheck
-rw-rw-r-- 1 paleti paleti 2.0K Oct  4 22:52 magiccheck.c
-rwxrwxr-x 1 paleti paleti  21K Oct  4 22:34 magicgenerate
-rw-rw-r-- 1 paleti paleti 9.2K Oct  4 23:19 magicgenerate.c
-rwxrwxr-x 1 paleti paleti  17K Oct  4 20:40 matrixmultiplication
-rw-rw-r-- 1 paleti paleti 1.5K Oct  4 20:39 matrixmultiplication.c
-rwxrwxr-x 1 paleti paleti  17K Oct  4 17:34 mergesort
-rw-rw-r-- 1 paleti paleti 1.7K Oct  4 17:34 mergesort.c
-rwxrwxr-x 1 paleti paleti  17K Oct  3 17:49 orphan
-rw-rw-r-- 1 paleti paleti  345 Oct  3 17:48 orphan.c
-rwxrwxr-x 1 paleti paleti  17K Oct  4 18:09 processcount
-rw-rw-r-- 1 paleti paleti  479 Oct  4 18:09 processcount.c
-rwxrwxr-x 1 paleti paleti  17K Oct  4 17:58 quicksort
-rw-rw-r-- 1 paleti paleti 1.4K Oct  4 17:57 quicksort.c
-rwxrwxr-x 1 paleti paleti  17K Oct  3 17:58 zombie
-rw-rw-r-- 1 paleti paleti  370 Oct  3 17:56 zombie.c

paleti@paletil:~$ !3
dir
ls -l
ls
```

```
paleti@paletil:~$ !5
ls -lh
dir
dir
ls -l
ls

paleti@paletil:~$ history

ls
ls -l
dir
dir
ls -lh
!3
!5
history

paleti@paletil:~$ quit
paleti@paletil:~/OS_LAB/fork_3$
```

Code : USING dup2 and pipes (IPC)

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <wait.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
```

Paleti Krishnasai
CED18I039

```c
#include <ctype.h>
#include <math.h>

void history(char his[], char cmd[])
{
   strcat(his, "\n");
   strcat(his, cmd);
}
int main()
{
   char his[1000] = "";
   char rec[1];
   char cmd[100] = {0};
   int top = 0;
   int fd[2];
   while (1)
   {
       pipe(fd);
       printf("%s", "\npaleti@paletil:~$");
       scanf("%[^\n]%*c", cmd);
       history(his, cmd);
       if (strcmp(cmd, "quit") == 0)
       {
           break;
       }
       char arg[10][100] = {0};
       int argc = 0, count = 0;
       for (int i = 0; i < strlen(cmd); i++)
       {
           if (cmd[i] == ' ')
           {
               argc++;
               count = 0;
           }
           else
           {
               arg[argc][count++] = cmd[i];
```

```
            }
        }
        char *argv[10] = {0};
        int k = 0;
        for (k = 0; k <= argc; k++)
        {
            argv[k] = arg[k];
        }
        argv[k] = NULL;
        int pid = fork();
        if (pid == 0)
        {
            if (!strcmp(cmd, "history"))
            {
                printf("%s\n", his);
            }
            else
            {
                dup2(fd[1], 1);
                execvp(*argv, argv);
            }
            exit(0);
        }
        else
        {
            wait(NULL);
            close(fd[1]);
            while (read(fd[0], rec, 1) > 0)
            {

                printf("%c", rec[0]);
            }
            close(fd[0]);
        }
    }
}
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/fork_3$ ./cmd_2

paleti@paletil:~$ls
cmd
cmd_2
cmd_2.c
cmd.c
fork-3
histogram
histogram.c
magiccheck
magiccheck.c
magicgenerate
magicgenerate.c
matrixmultiplication
matrixmultiplication.c
mergesort
mergesort.c
orphan
orphan.c
processcount
processcount.c
quicksort
quicksort.c
zombie
zombie.c

paleti@paletil:~$ls -l
total 1408
-rwxrwxr-x 1 paleti paleti   17160 Oct  4 22:02 cmd
-rwxrwxr-x 1 paleti paleti   17384 Oct  4 22:04 cmd_2
-rw-rw-r-- 1 paleti paleti    1646 Oct  4 22:04 cmd_2.c
-rw-rw-r-- 1 paleti paleti    1369 Oct  4 22:02 cmd.c
-rw-rw-r-- 1 paleti paleti 1166882 Sep 16 12:13 fork-3
-rwxrwxr-x 1 paleti paleti    8624 Sep 22 17:19 histogram
-rw-rw-r-- 1 paleti paleti    1101 Sep 22 17:19 histogram.c
-rwxrwxr-x 1 paleti paleti   16960 Oct  4 22:52 magiccheck
-rw-rw-r-- 1 paleti paleti    2028 Oct  4 22:52 magiccheck.c
-rwxrwxr-x 1 paleti paleti   21432 Oct  4 22:34 magicgenerate
-rw-rw-r-- 1 paleti paleti    9375 Oct  4 22:33 magicgenerate.c
-rwxrwxr-x 1 paleti paleti   16976 Oct  4 20:40 matrixmultiplication
-rw-rw-r-- 1 paleti paleti    1505 Oct  4 20:39 matrixmultiplication.c
-rwxrwxr-x 1 paleti paleti   17064 Oct  4 17:34 mergesort
```

```
-rwxrwxr-x 1 paleti paleti   17064 Oct  4 17:34 mergesort
-rw-rw-r-- 1 paleti paleti    1675 Oct  4 17:34 mergesort.c
-rwxrwxr-x 1 paleti paleti   16776 Oct  3 17:49 orphan
-rw-rw-r-- 1 paleti paleti     345 Oct  3 17:48 orphan.c
-rwxrwxr-x 1 paleti paleti   16784 Oct  4 18:09 processcount
-rw-rw-r-- 1 paleti paleti     479 Oct  4 18:09 processcount.c
-rwxrwxr-x 1 paleti paleti   17040 Oct  4 17:58 quicksort
-rw-rw-r-- 1 paleti paleti    1428 Oct  4 17:57 quicksort.c
-rwxrwxr-x 1 paleti paleti   16816 Oct  3 17:58 zombie
-rw-rw-r-- 1 paleti paleti     370 Oct  3 17:56 zombie.c

paleti@paletil:~$dir
cmd      histogram      magicgenerate.c         orphan          quicksort.c
cmd_2    histogram.c    matrixmultiplication    orphan.c        zombie
cmd_2.c  magiccheck     matrixmultiplication.c  processcount    zombie.c
cmd.c    magiccheck.c   mergesort               processcount.c
fork-3   magicgenerate  mergesort.c            quicksort

paleti@paletil:~$history

ls
ls -l
dir
history

paleti@paletil:~$exit

paleti@paletil:~$quit
paleti@paletil:~/OS_LAB/fork_3$
```

5.) Develop a multiprocessing version of histogram generator to count occurrences of various characters in a given text.

Theory :

Storing the input in an array whose size is limited by the 128 ASCII characters and mapping the indexes to characters and the values as the count through typecasting, and printing in the parent process . **#** is used for visualization.

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Paleti Krishnasai
CED18I039

```c
int main()
{
   char input[512];
   while(1)
   {
       printf("Enter the input : ");
       scanf("%s",input);
       int count[128] ;
       for(int i =0; i<128;i++)
       {
           count[i]=0;
       }

       pid_t pid;
       pid = vfork();

       if(pid == 0)
       {
           for(int i = 0;i<strlen(input);i++)
           {
               count[(int)input[i]]++;
           }
           exit(0);
       }
       else if (pid > 0)
       {
           wait(NULL);
           for(int i = 0;i<128;i++)
           {
               printf("%c --> %d",(char)i,count[i]);
               for(int j=0;j<count[i];j++)printf("#");
               printf("\n");
           }
           printf("\n");
           int flag;
           printf("Do you want to exit: 1-exit 0-repeat ");
```

Paleti Krishnasai
CED18I039

```
            scanf("%d",&flag);
            if(flag == 1)
            {
                return 0;
            }

        }
    }
    return 0;
}
```

```
paleti@paletil:~/OS_LAB/fork_3$ make histogram
make: 'histogram' is up to date.
paleti@paletil:~/OS_LAB/fork_3$ ./histogram
Enter the input : qqwer@$%$#EG
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
        --> 0

 --> 0

 --> 0

 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 --> 0
 0
 --> 0
 --> 0
 --> 0
 --> 0
  --> 0
! --> 0
" --> 0
```

Paleti Krishnasai
CED18I039

```
" --> 0
# --> 1#
$ --> 2##
% --> 1#
& --> 0
' --> 0
( --> 0
) --> 0
* --> 0
+ --> 0
, --> 0
- --> 0
. --> 0
/ --> 0
0 --> 0
1 --> 0
2 --> 0
3 --> 0
4 --> 0
5 --> 0
6 --> 0
7 --> 0
8 --> 0
9 --> 0
: --> 0
; --> 0
< --> 0
= --> 0
> --> 0
? --> 0
@ --> 1#
A --> 0
B --> 0
C --> 0
D --> 0
E --> 1#
F --> 0
G --> 1#
H --> 0
I --> 0
J --> 0
K --> 0
L --> 0
```

Paleti Krishnasai
CED18I039

```
L --> 0
M --> 0
N --> 0
O --> 0
P --> 0
Q --> 0
R --> 0
S --> 0
T --> 0
U --> 0
V --> 0
W --> 0
X --> 0
Y --> 0
Z --> 0
[ --> 0
\ --> 0
] --> 0
^ --> 0
_ --> 0
` --> 0
a --> 0
b --> 0
c --> 0
d --> 0
e --> 1#
f --> 0
g --> 0
h --> 0
i --> 0
j --> 0
k --> 0
l --> 0
m --> 0
n --> 0
o --> 0
p --> 0
q --> 2##
r --> 1#
s --> 0
t --> 0
u --> 0
v --> 0
```

```
q --> 2##
r --> 1#
s --> 0
t --> 0
u --> 0
v --> 0
w --> 1#
x --> 0
y --> 0
z --> 0
{ --> 0
| --> 0
} --> 0
~ --> 0
 --> 0

Do you want to exit: 1-exit 0-repeat
```

Paleti Krishnasai
CED18I039

6.) Develop a multiprocessing version of the matrix multiplication

Theory :
In the code , the number of rows of the second matrix is equal to the number of columns of the first matrix , hence we do not take the input in that case to ensure legal inputs from users.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>


int main()
{
   int n1, n2, n3;
   printf("Enter no. of rows of first matrix: ");
   scanf("%d", &n1);

   printf("Enter no. of rows of second matrix: ");
   scanf("%d", &n2);

   printf("\n Enter no. of columns of second matrix: ");
   scanf("%d", &n3);

   int i, j, k;

   int a[n1][n2], b[n2][n3], c[n1][n3];

   for (i = 0; i < n1; i++)
   {
       for (j = 0; j < n3; j++)
       {
           c[i][j] = 0;
       }
```

```
}

printf("Enter matrix a \n");

for (i = 0; i < n1; i++)
{
    for (j = 0; j < n2; j++)
    {
        scanf("%d", &a[i][j]);
    }
}

printf("\n");

printf("Enter matrix b \n");

for (i = 0; i < n2; i++)
{
    for (j = 0; j < n3; j++)
    {
        scanf("%d", &b[i][j]);
    }
}

pid_t child;

for (i = 0; i < n1; i++)
{
    for (j = 0; j < n3; j++)
    {
        child = vfork();
        if (child == 0)
        {
            for (k = 0; k < n2; k++)
            {
                c[i][j] += a[i][k] * b[k][j];
            }
```

```
                exit(0);
            }
        }
    }

    printf("\nProduct of the two matrices is \n");

    for (i = 0; i < n1; i++)
    {
        for (j = 0; j < n3; j++)
        {
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/fork_3$ ./matrixmultiplication
Enter no. of rows of first matrix: 3
Enter no. of rows of second matrix: 3

 Enter no. of columns of second matrix: 3
Enter matrix a
1 1 3
3 2 5
9 8 6

Enter matrix b
9 8 6
1 1 3
3 2 5

Product of the two matrices is
19 15 24
44 36 49
107 92 108
paleti@paletil:~/OS_LAB/fork_3$ ./matrixmultiplication
Enter no. of rows of first matrix: 3
Enter no. of rows of second matrix: 4

 Enter no. of columns of second matrix: 4
Enter matrix a
1 1 2 3
5 6 8 9
2 5 7 8

Enter matrix b
1 1 1 2
2 2 2 2
3 6 9 8
5 4 8 1

Product of the two matrices is
24 27 45 23
86 101 161 95
73 86 139 78
```

Paleti Krishnasai
CED18I039

EXTRA CREDITS :

7.) Develop a parallelized application to check for if a user input square matrix is a magic square or not , extend the code to support magic square generation (input will be order of matrix)

Theory :

In any magic square, the first number i.e. 1 is stored at position (n/2, n-1). Let this position be (i,j). The next number is stored at position (i-1, j+1) where we can consider each row & column as circular array i.e. they wrap around.

Three conditions hold:
1. The position of next number is calculated by decrementing row number of the previous number by 1, and incrementing the column number of the previous number by 1. At any time, if the calculated row position becomes -1, it will wrap around to n-1. Similarly, if the calculated column position becomes n, it will wrap around to 0.
2. If the magic square already contains a number at the calculated position, calculated column position will be decremented by 2, and calculated row position will be incremented by 1.
3. If the calculated row position is -1 & calculated column position is n, the new position would be: (0, n-2).

Code :

MAGIC SQUARE CHECK

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Returns 1 if mat[][] is magic
// square, else returns 0.
int isMagicSquare(int *mat, int N)
{
   // calculate the sum of
   // the prime diagonal
```

Paleti Krishnasai
CED18I039

```
int sum = 0, sum2 = 0;

int diag_pid = vfork();
if (diag_pid == 0)
{
    for (int i = 0; i < N; i++)
        sum = sum + *((mat + i * N) + i);
    exit(0);
}
else if (diag_pid > 0)
{
    wait(NULL);
    // the secondary diagonal
    for (int i = 0; i < N; i++)
        sum2 = sum2 + *((mat + i * N) + (N - 1 - i));

    if (sum != sum2)
        return 0;
}

int row_pid = vfork();
if (row_pid == 0)
{
    // For sums of Rows
    for (int i = 0; i < N; i++)
    {

        int rowSum = 0;
        for (int j = 0; j < N; j++)
            rowSum += *((mat + i * N) + j);

        // check if every row sum is
        // equal to prime diagonal sum
        if (rowSum != sum)
            return 0;
    }
    exit(0);
```

```
    }
    else if (row_pid > 0)
    {
        wait(NULL);
        // For sums of Columns
        for (int i = 0; i < N; i++)
        {

            int colSum = 0;
            for (int j = 0; j < N; j++)
                colSum += *((mat + j * N) + i);

            // check if every column sum is
            // equal to prime diagonal sum
            if (sum != colSum)
                return 0;
        }
    }

    return 1;
}

int main()
{
    int n, i, j;

    printf("Enter order of matrix:-\n");
    scanf("%d", &n);

    int A[n][n];
    printf("Enter matrix:-\n");

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &A[i][j]);
```

```
        }
    }



    if (isMagicSquare((int *)A, n))
    {
        printf("Magic Square\n");
    }
    else if(isMagicSquare)
    {
        printf("Not a Magic Sqaure\n");

    }
    return 0;
}
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/fork_3$ ./magiccheck
Enter order of matrix:-
3
Enter matrix:-
2 7 6
9 5 1
4 3 8
Magic Square
paleti@paletil:~/OS_LAB/fork_3$ ./magiccheck
Enter order of matrix:-
4
Enter matrix:-
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
Magic Square
paleti@paletil:~/OS_LAB/fork_3$ 5
5: command not found
paleti@paletil:~/OS_LAB/fork_3$ ./magiccheck
Enter order of matrix:-
5
Enter matrix:-
9 3 22 16 15
2 21 20 14 8
25 19 13 7 1
18 12 6 5 24
11 10 4 23 17
Magic Square
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/fork_3$ ./magiccheck
Enter order of matrix:-
3
Enter matrix:-
1 2 3
3 2 1
2 1 3
Not a Magic Sqaure
paleti@paletil:~/OS_LAB/fork_3$ ./magiccheck
Enter order of matrix:-
4
Enter matrix:-
1 2 3 4
3 2 1 4
2 1 3 4
4 1 2 3
Not a Magic Sqaure
paleti@paletil:~/OS_LAB/fork_3$
```

```
paleti@paletil:~/OS_LAB/fork_3$ ./magiccheck
Enter order of matrix:-
2
Enter matrix:-
2 3
3 2
Not a Magic Sqaure
```

Paleti Krishnasai
CED18I039

Theory :
referred http://www.1728.org/magicsq3.htm and GeeksForGeeks for magic square generation .

Code :

GENERATION

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <unistd.h>

#define DEFAULT 100

void odd_order(int n, int a[][DEFAULT])
{
   int i = n / 2;
   int j = n - 1;
   int p = n / 2;
   int q = 0;
   int split = ((n * n) / 2);
   pid_t pid1, pid2;
   pid1 = fork();
   if (pid1 == 0)
   {
       for (int num = 1; num <= split + 1;)
       {
           if ((i == -1) && (j == n))
           {
               i = 0;
               j = n - 2;
           }
           else
           {
```

```
            if (j == n)
            {
                j = 0;
            }
            if (i < 0)
            {
                i = n - 1;
            }
        }
        if (a[i][j])
        {
            j -= 2;
            ++i;
            continue;
        }
        else
        {
            a[i][j] = num;
            ++num;
        }
        ++j;
        --i;
    }
    exit(0);
}
else if (pid1 > 0)
{
    pid2 = fork();
    if (pid2 == 0)
    {
        for (int num = n * n; num > split + 1;)
        {
            if ((p == n) && (q == -1))
            {
                p = 0;
                q = n - 2;
            }
```

```
            else
            {
                if (q == -1)
                {
                    q = n - 1;
                }
                if (p > n - 1)
                {
                    p = 0;
                }
            }
            if (a[p][q])
            {
                q += 2;
                --p;
                continue;
            }
            else
            {
                a[p][q] = num;
                --num;
            }
            --q;
            ++p;
        }
        exit(0);
    }
    else if (pid2 < 0)
    {
        printf("\nForking failed\n");
        exit(0);
    }
}
else
{
    printf("\nForking failed\n");
    exit(0);
```

```
    }
    int status;
    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);
    return;
}

void singly_even_order(int n, int a[][DEFAULT])
{
    odd_order(n / 2, a);
    pid_t pid1, pid2, pid3;
    pid1 = fork();
    if (pid1 == 0)
    {
        for (int i = n / 2; i < n; ++i)
        {
            for (int j = n / 2; j < n; ++j)
            {
                a[i][j] = a[i - n / 2][j - n / 2] + (n * n / 4);
            }
        }
        exit(0);
    }
    else if (pid1 > 0)
    {
        pid2 = fork();
        if (pid2 == 0)
        {
            for (int i = 0; i < n / 2; ++i)
            {
                for (int j = n / 2; j < n; ++j)
                {
                    a[i][j] = a[i][j - n / 2] + (2 * n * n / 4);
                }
            }
            exit(0);
        }
```

```
    else if (pid2 > 0)
    {
        pid3 = fork();
        if (pid3 == 0)
        {
            for (int i = n / 2; i < n; ++i)
            {
                for (int j = 0; j < n / 2; ++j)
                {
                    a[i][j] = a[i - n / 2][j] + (3 * n * n / 4);
                }
            }
            exit(0);
        }
        else if (pid3 < 0)
        {
            printf("\nForking failed\n");
            exit(0);
        }
    }
    else
    {
        printf("\nForking failed\n");
        exit(0);
    }
}
else
{
    printf("\nForking failed\n");
    exit(0);
}
int status;
waitpid(pid1, &status, 0);
waitpid(pid2, &status, 0);
waitpid(pid3, &status, 0);
int count;
pid1 = fork();
```

```
if (pid1 == 0)
{
    for (int i = 0; i < n / 2; ++i)
    {
        int j = -1;
        if (i == n / 4)
        {
            ++j;
        }
        count = n / 4;
        for (; count > 0; ++j, --count)
        {
            int temp = a[i][j + 1];
            a[i][j + 1] = a[i + n / 2][j + 1];
            a[i + n / 2][j + 1] = temp;
        }
    }
    exit(0);
}
else if (pid1 > 0)
{
    pid2 = fork();
    if (pid2 == 0)
    {
        count = n / 4 - 1;
        while (count > 0)
        {
            for (int i = 0; i < n / 2; ++i)
            {
                int temp = a[i][n - count];
                a[i][n - count] = a[i + n / 2][n - count];
                a[i + n / 2][n - count] = temp;
            }
            --count;
        }
        exit(0);
    }
```

```
        else if (pid2 < 0)
        {
            printf("\nForking failed\n");
            exit(0);
        }
    }
    else
    {
        printf("\nForking failed...\n");
        exit(0);
    }

    waitpid(pid1, &status, 0);
    waitpid(pid2, &status, 0);
    return;
}

void doubly_even_order(int n, int a[][DEFAULT])
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            a[i][j] = (n * i) + j + 1;
        }
    }
    pid_t TLeft, TRight, BLeft, BRight, center;
    TLeft = fork();
    if (TLeft == 0)
    {
        for (int i = 0; i < n / 4; ++i)
        {
            for (int j = 0; j < n / 4; ++j)
            {
                a[i][j] = (n * n + 1) - a[i][j];
            }
        }
```

```
        exit(0);
    }
    else if (TLeft > 0)
    {
        TRight = fork();
        if (TRight == 0)
        {
            for (int i = 0; i < n / 4; ++i)
            {
                for (int j = 3 * (n / 4); j < n; ++j)
                {
                    a[i][j] = (n * n + 1) - a[i][j];
                }
            }
            exit(0);
        }
        else if (TRight > 0)
        {
            BLeft = fork();
            if (BLeft == 0)
            {
                for (int i = 3 * (n / 4); i < n; ++i)
                {
                    for (int j = 0; j < n / 4; ++j)
                    {
                        a[i][j] = (n * n + 1) - a[i][j];
                    }
                }
                exit(0);
            }
            else if (BLeft > 0)
            {
                BRight = fork();
                if (BRight == 0)
                {
                    for (int i = 3 * (n / 4); i < n; ++i)
                    {
```

```
                    for (int j = 3 * (n / 4); j < n; ++j)
                    {
                        a[i][j] = (n * n + 1) - a[i][j];
                    }
                }
                exit(0);
            }
            else if (BRight > 0)
            {
                center = fork();
                if (center == 0)
                {
                    for (int i = (n / 4); i < 3 * (n / 4); ++i)
                    {
                        for (int j = n / 4; j < 3 * (n / 4); ++j)
                        {
                            a[i][j] = (n * n + 1) - a[i][j];
                        }
                    }
                    exit(0);
                }
                else if (center < 0)
                {
                    printf("\nForking failed\n");
                    exit(0);
                }
            }
            else
            {
                printf("\nForking failed\n");
                exit(0);
            }
        }
        else
        {
            printf("\nForking failed\n");
            exit(0);
```

```
            }
        }
        else
        {
            printf("\nForking failed\n");
            exit(0);
        }
    }
    else
    {
        printf("\nForking failed\n");
        exit(0);
    }
    int status;
    waitpid(TLeft, &status, 0);
    waitpid(TRight, &status, 0);
    waitpid(BLeft, &status, 0);
    waitpid(BRight, &status, 0);
    waitpid(center, &status, 0);
    return;
}

void print(int n, int a[][DEFAULT])
{
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int n;
```

```
printf("\nEnter the order of Magic square : ");
scanf("%d", &n);
int shm_id;
key_t key = IPC_PRIVATE;
shm_id = shmget(key, sizeof(int[n][n]), IPC_CREAT | 0666);
int(*a)[n];
if (shm_id < 0)
{
    printf("\nSHM Failed\n");
    exit(0);
}
a = shmat(shm_id, 0, 0);
if (a == (void *)-1)
{
    printf("\nSHM Failed\n");
    exit(0);
}
if ((n <= 0) || (n == 2))
{
    printf("\nMagic square not possible\n");
}
else if (n == 1)
{
    printf("\nMagic square of Order %d\n", n);
    printf("1\n");
}
else if (n % 2 == 1)
{
    printf("\nMagic square of Order %d\n", n);
    odd_order(n, a);
    print(n, a);
}
else if (n % 4 == 0)
{
    printf("\nMagic square of Order %d\n", n);
    doubly_even_order(n, a);
    print(n, a);
```

```
    }
    else
    {
        printf("\nMagic square of Order %d\n", n);
        singly_even_order(n, a);
        print(n, a);
    }
    if (shmdt(a) == -1)
    {
        exit(0);
    }
    if (shmctl(shm_id, IPC_RMID, NULL) == -1)
    {
        exit(0);
    }
    return 0;
}
```

Paleti Krishnasai
CED18I039

```
paleti@paletil:~/OS_LAB/fork_3$ ./magicgenerate

Enter the order of Magic square : 3

Magic square of Order 3
2 7 6
9 5 1
4 3 8
paleti@paletil:~/OS_LAB/fork_3$ ./magicgenerate

Enter the order of Magic square : 4

Magic square of Order 4
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
paleti@paletil:~/OS_LAB/fork_3$ ./magicgenerate

Enter the order of Magic square : 5

Magic square of Order 5
9 3 22 16 15
2 21 20 14 8
25 19 13 7 1
18 12 6 5 24
11 10 4 23 17
paleti@paletil:~/OS_LAB/fork_3$
```